RICE UNIVERSITY

Tracking Evolution of Learning on a Visualmotor Task

by

Sameer Siruguri

A Thesis Submitted
in Partial Fulfillment of the
Requirements for the Degree

Master of Science

Approved, Thesis Committee:

Devika Subramanian, Associate
Professor, Chair
Computer Science

Moshe Vardi, George Professor,
Department Chair,
Computer Science

Dan Osherson, Autry Professor,
Psychology

Houston, Texas

May, 2001

(blank)

ABSTRACT

Tracking Evolution of Learning on
a Visualmotor Task
by
Sameer Siruguri

We construct models of the evolution of human learning on a visualmotor task by analysing a large sequential corpus of low-level peformance data generated from it. The performance data is drawn sparsely from a large, high-dimensional space, is non-stationary—slowly evolving control policies are punctuated by radical conceptual shifts—and has non-Gaussian noise, which is difficult to model.

We develop novel, data-driven algorithms for identifying the conceptual shifts, and for constructing compact representations of the subjects' stationary control policies. The policy models are "local" and use a novel extension to locally weighted regression. The closeness of fit of model performance to human learning curves experimentally demonstrates the effectiveness of our methods. In contrast to previous modeling work, we make no *a priori* assumptions about the underlying cognitive architecture required to duplicate subject behavior. By comparing the performance of our methods to decision trees, we demonstrate the superiority of local models for learning compact representations of high-dimensional, noisy, non-stationary sequential data.

(blank)

Acknowledgments

I would like to thank my advisor, Dr Devika Subramanian, without whose guidance and insightful comments, this thesis would have not gotten this far. I am immeasurably indebted to my family and friends without whom *I* wouldn't have gotten this far, especially those who have lent a sympathetic shoulder to my frequent complaints. Without all the affection (and other stimulants) they plied me with, inspiration would have long run dry.

(blank)

Contents

Illustrations

(blank)

Tables

(blank)

Chapter 1

Introduction

This thesis describes methods for modeling the evolution of human learning on a particular visualmotor task. The task, which we shall also refer to as the *game*, involves directing a (point) submarine across a field of circular mines to within some distance of a rendezvous point in a bounded amount of time. We analyse this task through data generated during play of the game: at each time step, the values of a suite of environment indicators as well as the player response are recorded.

The aim of explaining the evolution of learning is to *bootstrap* the process of acquiring the skills required to perform the task. To do this, we first obtain subject data that represents the interaction of our subjects with the task at hand. This is done by conducting subject sessions on the task according to a data-gathering protocol designed to generate enough data to make the modeling task feasible without exhausting the subjects. Our aim is to find algorithms that will analyse this data and construct models of the learning exhibited during the generation of this data that can expose the subject's inadequacies and thus help correct them.

Using a model of subjects' learning is necessary to produce didactic feedback for them because teaching this game using a classroom drill is very difficult. There is no single optimal policy for this task; each player learns a different strategy that depends on his peculiar hand-eye coordination skills and on his abilities to use the visual display of the task to understand which aspects of the task are key to performing well in it. Hence, instead of teaching a textbook "super-strategy", we want to understand, from the visualmotor data generated by the subjects, what aspects of the game they have learnt and what aspects remain to be discovered. We construct models of subjects who have played the task for a while to help understand what the best methods for discovering these facts are. We emphasise here that we construct these models, not to extract any expert strategies that would be force-fed to new players, but to understand which method is best at dynamically constructing models that can bootstrap a new player's skills.

Building these models is a challenge because of the following properties of the task and the data generated:

1. Information about subject strategies cannot be obtained from the subjects themselves. Verbalisation during task performance impairs a subject's ability to acquire the task. Further, it is difficult for humans to introspect on their learning after acquiring the task—their analyses tend to be incomplete and/or inaccurate.

2. Learning on this task consists of two components, which we will refer to as the *reactive* and the *planning* components. The former entails learning reflex actions for specific situations in the game. The latter requires the formation of global plans that solve classes of situations and rules for transiting from one high-level plan to another. This includes understanding the long-term consequences of local, reactive decisions as well as learning to cope with the limited view that the interface to the game affords the subject.

   These components evolve simultaneously but might not do so in a synchronised fashion: the high-level plans might remain the same while their visualmotor implementation—the reactive decisions—change and vice-versa. One challenge for a strategy model, therefore, is to explain the possibly uncoordinated co-evolution of the two components.

3. The data being modeled is non-stationary because the subjects we are modeling improve on performance over time. This means that the patterns in the data that correspond to their strategies will change over time. Since data-mining methods extract patterns by testing for the relative likelihoods of different candidates in the pattern space, a pattern is most reliably detected in given data, if it is the most likely pattern in that data. If we treat non-

stationary data as stationary, more than one pattern becomes equally likely in the data, each pattern being the dominant one in its corresponding stationary portion of the data stream, which we refer to as a *learning plateau.* This makes pattern extraction difficult. Hence, another challenge in this problem is to pinpoint the location and length of the learning plateaus in the data as accurately as possible. We will refer to points between learning plateaus, where the pattern distributions change, as *changepoints.* Changepoints may be *sharp*, meaning that pattern distributions change abruptly, or *wide*, meaning that pattern distributions change slowly, with a relatively longer portion of the data stream containing a near-equal mixture of the stationary distributions on either side of the changepoint.

1.1 Methods for building models

Our methods for building models are motivated by the properties enumerated above. The first item motivates us to only use data-driven methods in our modeling efforts: we treat this as a data-mining problem, where the low-level percept-response data is our only ground truth. The subject sessions are conducted over a fixed length of time, which we call the subject's *training period.* Both components of learning—reflex acquisition and plan acquisition—are to be extracted by compressing the data generated during the training period and detecting patterns that correspond to the local and global components of the subject's strategy. The key is to find the right data-compression method, the one that will unambiguously reveal these patterns. In particular, we do not wish to specify *a priori* a task decomposition and the corresponding parameters of a cognitive architecture that reflect the strategy of the human. This is not only because the validity of an *a priori* decomposition is hard to verify but also because such a decomposition could easily be subject-specific. Online training of an arbitrary number of subjects will be impractical if an architectural specification of each subject's cognitive processes on the task was required first.

To understand what approaches can handle the second item, we will first define the two components of learning more formally. A response is said to come from the reactive component of the learned strategy if it is of the form $\pi : \mathbf{P} \rightarrow \mathcal{P}(\mathbf{A})$, where $\mathbf{P}$ is the set of possible readings of the environment indicators. The planning component of the learned strategy represents higher-level knowledge about the task by constructing a framework that defines transitions from one subset of π to another. These transitions can be arranged in a hierarchy to represent abstractions of the subsets into classes of situations faced in the task. At each point in time, the subject follows a path down this hierarchy, creating a stack of abstractions, the top of which contains a subset of π, from which a response to the current percept is generated. The hierarchy acts as a guide to the efficient generation of this response. Each abstraction a thereby represents a subgoal in the task, the solution to which is contained in any subset of π that can occur on the top of the abstraction stack generated after putting a on it.

To discuss how we can model this hierarchy, we write it formally as a set of directed labeled graphs, $\mathbf{G}$. Each node in a graph in $\mathbf{G}$ represents either a member of $\mathbf{G}$ or a subset of π. Each edge has two labels: one is a conditional expression that is evaluated on the percept history and the other is a number in $[0, 1]$, representing a probability of switching from one subgoal to another at some level in the subgoal stack, when the condition evaluates to true. The highest level graph contains a single node, which is simply a place-holder for the goal of solving the task. The conditional expressions represent the partitioning of π into subsets that the subject uses to construct the abstraction hierarchy. If multiple expressions are true at some point, the probabilities give the relative strengths with which the expressions decide the transition that is made. This hierarchical structure is static only when the subject is on a learning plateau. Hence, the subject strategy over the entire training period is actually a sequence of n such structures, where n is the number of changepoints in the training period of the subject. Each structure in the sequence evolves into the next one as the subject learns the task; the representation of this continuous evolution as a finite set of distinct structures is merely a convenient discretisation of this process.

The problem of understanding subject strategy and strategy evolution is, therefore, the problem of constructing this sequence of graph hierarchies. If we also understand what changes in the structure—addition of nodes and edges or modification of the transition conditionals and probabilities—result in the evolution of one hierarchy into the next, we will have also understood the reasons for the inadequacies of a particular strategy.

We approach the problem of creating these strategy models as a two-step one:

1. First, duplicate the performance of subjects who have already played the game according to a well-defined training protocol. We are, in other words, constructing a model that is behaviorally equivalent to the subject. We define behavioral equivalence between two strategies as a match between their *performances* on the task, where performance is defined as the function, $S_W(e) = \text{success}(e) - \text{success}(e - W)$, $W < e \leq N$, where N is the total number of episodes played by the two strategies, success(e) is the number of successes in episodes 0 through e and W is some constant. A model that satisfies this criterion will be called a *representative model.* In building a representative model, we are not concerned with its cognitive plausibility. Since the game has two other possible outcomes, an explosion or a timeout, we can also define closeness by counting the number of explosions or timeouts, rather than successes, over an episode window of size W, or by combining more than one of these counts.

   This thesis discusses this half of the problem: developing a language, that is, a set of terminal and non-terminal symbols and grammar rules for parsing symbol strings, that can be used to describe a representative model. We will also develop ways to measure which language is best for expressing representative models in and which representative model expressed in a particular language is adequate for the purposes of duplicating the performance level of a human strategy.

2. Second, use the methods developed for constructing representative models

from expert data to build models that will bootstrap the skills of subjects whose performance at the task is being monitored. Unlike representative models, these models are intended to adaptively generate training protocols for the monitored subjects that will improve their skills at the task. Therefore, these models should be easily understood by humans, which means they should be simple and compact. A model that is most easily communicated to humans will be called an *explanatory model.*

1.2 Approach in the Thesis

My approach in this thesis will be to first concentrate on developing a representative model. A language that is easy to write representative models in might be poorly suited to write explanatory models in. We will, however, begin with representative models because there is a simple objective evaluation measure for such models. Also, we expect that a model that can duplicate subject performance will give some insight into the nature of the strategies that the subject has learnt as well the nature of good explanatory models. If the representative model turns out also to be an explanatory one, then both the above aims will be met. If not, we will then attempt either to find a method for simplifying the representative model into an explanatory model or to develop a explanatory model independently. In the first case, the language used to describe the two models will be the same. In the second case, the representative model might motivate the development of the explanatory model in some way but the language will be a different, simpler one.

The unique contribution of this thesis is the construction of methods that take into account the two components of learning and that also handle the non-stationary nature of data. Previous work has concentrated on only one of the components of learning: either tracking skill acquisition or plan acquisition but not both. The data used to build models has also been stationary: the subject's strategy is allowed to stabilise and only then is a model of subject performance constructed.

In this thesis, we present a novel method for estimating the location of changepoints in data in order to tackle non-stationarity. The method is based on the idea of developing a *strategy derivative*, a function that measures the rate of change of the strategy with respect to time. We also discuss the representative model we have developed for the game. This method maintains a low-level description of the strategy which allows us to easily analyse the evolution of its reactive component. We analyse the planning component by showing that certain transformations of the low-level description perform better than others, which we consider evidence for our hypothesis that the subject has similarly abstracted information from sets of reactive decisions into high-level goal-based knowledge.

We develop the representative model by arguing for the use of “local” methods as a better alternative to “global” methods for finding such models. Local models dynamically vary the importance of portions of the input space as the query to the model varies and use this information to calculate a response to the query. We show the superior performance of our local models over that of decision trees, a global model commonly used for classification and rule inference. We describe a novel extension [SS01b][SS01a] to an algorithm used as the response calculation component of local methods and compare the results obtained from using this extension versus those obtained from using traditional methods.

In the next chapter, I will describe the game and the data collection protocol in greater detail. I will then introduce formal notation that underlie the methods we use for analysing the data. In the third chapter, I will discuss the the changepoint detection method, the representative model that we have formulated for this task and the results obtained on evaluating this model. I will then compare this model with previous work in this field and conclude with a discussion of our efforts to construct explanatory models for the game.

Chapter 2

A Formal Model

This chapter is organised as follows. In the first section of this chapter, we describe the submarine task in greater detail. In the rest of the chapter, we will discuss the formal notation that we will use for the models of human learning we are interested in constructing. We will motivate this notation by first presenting a mathematical description of subject strategy on the task. This description is, in turn, motivated by the analysis of the components of learning that we have carried out in the previous chapter.

We then present some characteristics of the data that are central to the problem of model construction and discuss why these characteristics point to the importance of using "local" rather than "global" methods for solving this problem. We also discuss the nature of a measure used to characterise the data and how the properties of this measure motivate the development of the representative model that we have actually used.

2.1 The Submarine Task

The game outlined in the previous chapter is the NRL Submarine Task[GS96], developed by Alan Schultz at the Naval Research Laboratory. It involves directing a (point) submarine across a field of circular mines to within some distance of a rendezvous point in a bounded amount of time. The game ends if and only if either the rendezvous is reached, the submarine runs into a mine or time runs out; when this happens, an *episode* of the game is said to have ended. At each time step, the subject is given information about the immediate environs of the submarine through a panel display, to which a response, consisting of a choice of turn and speed, is given through a joystick. The panel display contains ten elements, with their ranges and increments given in Table 2.1. The continuous output of the joystick is discretised. The discretisation is shown in Table 2.2. The data collected is thus in the form of the values of the panel display and the corresponding response for each time step of the game. We will refer to the set of values of the panel display at a given time step as the *panel reading* or *input vector* and the combined response recorded for a given input vector as the *action vector* or *output vector.*

The data that we analyse in this thesis was collected for five subjects, who we will refer to as Subjects 1 through 5. Five undergraduates at San Diego State University participated in this experiment in 1996. The data collection was performed by our collaborators Diana Gordon and Sandra Marshall. The training period for the subjects varied from about 600 to about 1000 episodes (Table 3.3.1). An episode can go to a maximum of 200 time steps. The episodes are not identical; first, the submarine starts each episode at a randomly chosen point 1100 units away from the goal and second, the layout of the mines is different in each, being chosen from a uniform distribution over the portion of the plane where mines can be laid. This portion starts some distance from the submarine's start position, usually about 400 units later. It ends about 300-400 units before the goal is reached. We refer to this portion of the plane to which the mines are constrained as the *minefield.* For each subject, we denote by $\mathbf{E}$, the interval $[0, N-1]$ over $\mathbb{N}$, where N is the number of episodes that that subject played.

2.2 Specifying the Subject Strategy

As mentioned earlier, we will introduce a more rigorous description of subject strategy in order to segue into the formal notation for our models. This strategy description is for a stationary strategy between two changepoints in the training period and therefore does not take into account the length of training time elapsed.

Our description must reflect the division of the strategy into the two components discussed in the previous chapter. However, notating the planning component requires an explicit representation of the subgoal structure, which means we have to stipulate a task decomposition that the subject is using, an approach that we have decided against. At the same time, we cannot ignore the planning component altogether. We choose, therefore, to model this component by updating internal state that, at any given point of time, contains information about the aspects of the task that are deemed important for solving it. A change in the subgoals is thus reflected in a change in state, i.e. when subjects change a high-level plan in their strategies, they do so by paying attention to some previously neglected aspect of the task or by differently encoding the information they already consider important. We expect, therefore, that if the evolution of state representation is adequately accounted for, we will model the planning component of the subject strategy.

This analysis of strategy decomposition motivates the following notation for the subject strategy. Given an input vector $p \in \mathbf{P}$, the set of all possible input vectors, we assume that the subject computes an action vector a from $\mathbf{A}$, the set of all action vectors, using some action choice *policy* that utilises and updates the subject's internal state, taken from the set of possible state configurations, $\mathbf{S}$, for the task. That is, the policy is a mapping $\pi : P \times S \to A \times S$. This policy is the mathematical description of the subject's strategy that we have discussed in the previous chapter and henceforth, we will use the terms "policy" and "strategy" interchangeably. Hence, for each $e \in \mathbf{E}$, we have a sequence $\mathbf{R}_e = \{\langle p_0, \alpha_0, s_0 \rangle, \ldots, \langle p_{n_e}, \alpha_{n_e}, s_{n_e} \rangle\}$, where $p_i \in \mathbf{P}, \alpha_i \in \mathbf{A}, s_i \in \mathbf{S}$. This denotes, for each element $\langle p_i, \alpha_i \rangle$ in $\mathbf{R}_e$, that in episode e, the input p_i, coupled with the state value s_{i-1}, generates the response α_i and updates the state to s_i. The sequence $\{\mathbf{R}_0, \ldots, \mathbf{R}_{N-1}\}$ is the data generated during the training period of a subject and is called the subject's *data run*.

The validity of the use of a subject-independent strategy model is supported by the similarities observed in the learning curves of the participants of our data-gathering exercise. Figures 2.1 and 2.2 show learning for three subjects, who attain varying success rates at the end of five days of training. The curves are all characterised by a low-frequency general improvement in playing ability, punctuated with sudden increases in the success rate, mixed with high-frequency oscillations about

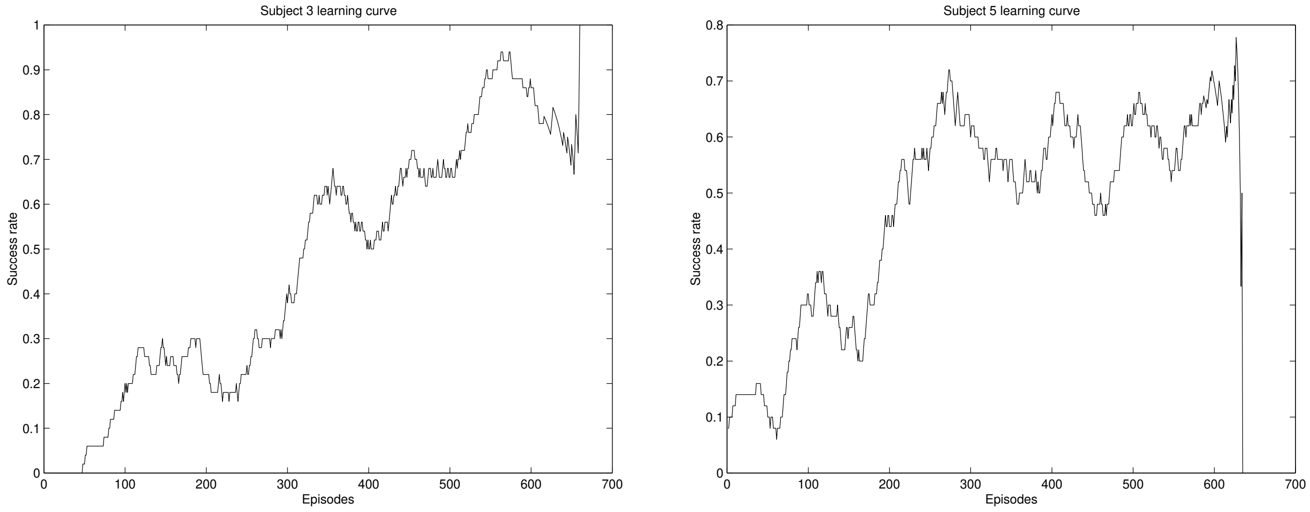

Figure 2.1 : (a) Subject 3 learning curve and (b) Subject 5 learning curve

Panel reading	Range	Step
Sonars (7)	$[10, 220]$	10
Bearing (1)	$[1, 12]$	1
Range (1)	$[100, 1100]$	100
Time left (1)	$[1, 200]$	1

Table 2.1 : Ranges of panel reading elements

the current proficiency level, observations corroborated by learning curves obtained from more recent experiments (which we do not show here.) Among our experimental subjects, we have two "experts," who reach a peak success rate of above 80%, two "intermediate" players who peak between 30% and 75% and a novice who fails to cross 10%. Further, we find, during construction of our explicit representations of their strategies, that players in the same proficiency category have similar representations, encouraging our use of a generic framework for modeling the multi-component strategies employed by them.

In the next section, we will use this description of the subject strategy to develop

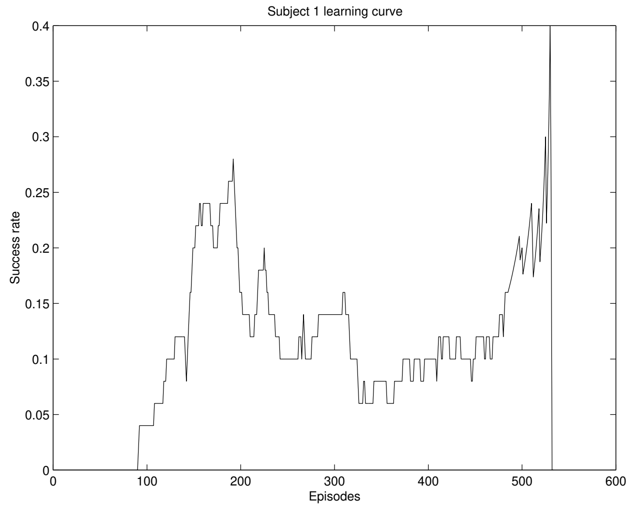

Figure 2.2 : Subject 1 learning curve

Action response	Range	Step
Turn	$[-32, 32]$	4
Speed	$[0, 40]$	5

Table 2.2 : Ranges of action responses

a similarly precise description of a generic representative model.

2.3 A Formal Model Specification

The model we have used to describe the subject strategy uses and updates internal state. *A priori*, the correct representation for state that most adequately reflects the subgoal structure is unknown. Hence, to begin with, we will specify a stateless model and develop state as a response to analyses of deficiencies in the performance of these models.

The stateless model will have one input, taken from $\mathbf{P}$, the set of all input vectors. One of the elements of the input vector is the amount of time left; hence, the model can track a change in policy caused by the human nearing the time deadline.

The model has a single action vector as its output, which is an action chosen using a distribution over the action space. We use a stochastic instead of a deterministic function for the action choice for two reasons. First, for a given input, there might be multiple "good" actions. These actions are all "good" in the sense that the model can choose any one of them to achieve the same performance. Hence, the model should be able to choose any one of these good actions. Second, the amount of state maintained by the subject might vary dynamically. This variation can be bursty, that is, the state space may change its dimensions for extremely short periods of time before reverting to the "default" dimensionality. If this happens, there won't be enough data to estimate state representation parameters so that a single action can be chosen reliably. In this case, we hope to calculate a distribution over actions that correctly marginalises the unknown parameters, yielding a mix of actions behaviourally equivalent to the subject's choice distribution.

We have, then, the model $M(p) = a$, where $M : \mathbf{P} \rightarrow \mathcal{P}_{(e,p)}(\mathbf{A})$, i.e. the model chooses an action $a \in \mathbf{A}$ according to the distribution $\mathcal{P}_{(e,p)}$, which itself depends on the value of e and p.

2.4 Partitioning the Input Space

In this section, we discuss some properties peculiar to the percept-response data generated by the submarine task. These properties motivate our choice for the kind of model, conforming to the notation discussed in Section 2.3, that we expect will be successful as representative models.

Since the percept-response data is the only manifestation we have of the underlying strategy, then, for a stateless model, information about the subject strategy is encapsulated in the response distribution in this data. Therefore, knowing what situations comprise subgoals in the subject strategy requires a distribution-based partitioning of the percept space. That is, we have to separate the percept space

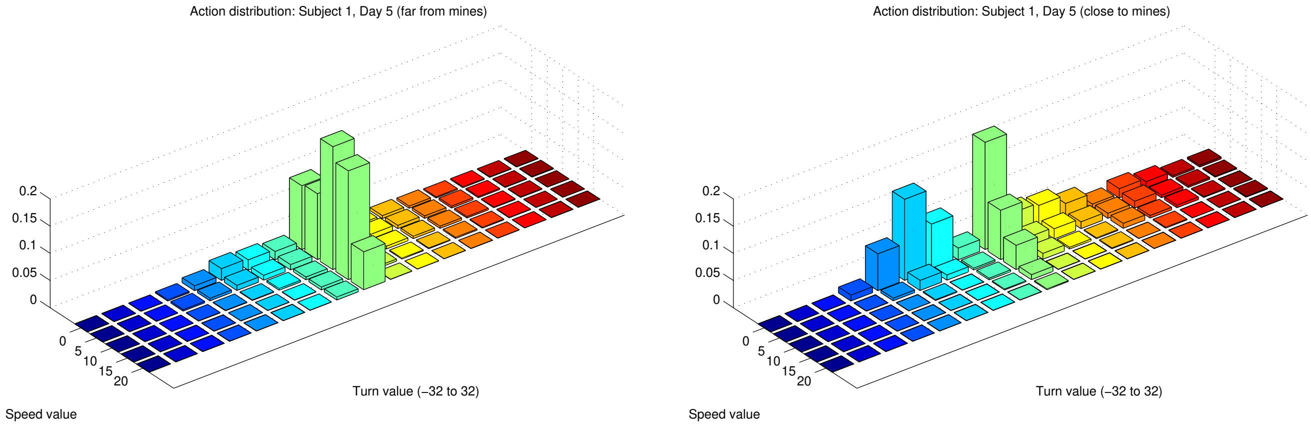

Figure 2.3 : Action distributions, Subject 1/Day 5, for (a) far-from-mine and (b) close-to-mine situations

into portions such that within a partition, the action distributions do not change with further splits but between two partitions, they are sufficiently different.

Note that if we incorporate state, the action distributions will have to be transformed to account for state information. For example, if we decide to represent state by keeping track of the last action, we simply get a new action distribution by transforming the input space to also contain the last action chosen for every panel reading. To begin with, we will form partitions based on the raw distributions, i.e. the distributions of actions over collections of the panel readings, without any state augmentation.

Figure 2.3 shows how distributions vary over disjoint subsets of $\mathbf{P}$, for a particular learning plateau for one of the subjects. The average of these two distributions is not representative of either and will perform poorly if used in either portion of the input space. The average also hides crucial aspects of the two that help describe how the subject's strategy changes with a change in the nature of the input. It is thus neither representative nor explanatory.

We define the partitioning idea more precisely *response-equivalent sets* by defining, for any two inputs to M, the property of response-equivalence:

Response-equivalence: Two inputs p and p' are said to be response-equivalent if the distributions $\mathcal{P}$ and $\mathcal{P}'$, from which the responses a and a' to p and p', respectively, are chosen, are such that $\Delta(\mathcal{P}, \mathcal{P}') < \epsilon$, for some small ϵ, where Δ is a function that measures the similarity between two probability distributions and $\forall \mathcal{P},\ \Delta(\mathcal{P}, \mathcal{P}) = 0$.

Define a relation R over $\mathbf{P} \times \mathbf{P}$, as pRp' iff p is response-equivalent to p'. Partition $\mathbf{P}$ over R to obtain a *response-equivalent partition* of $\mathbf{P}$. Given a response-equivalent partitioning R of $\mathbf{P}$, any further partitioning of the classes in R is also response-equivalent. In fact, creating as many partitions as there are members in $\mathbf{P}$ also gives a response-equivalent partition. However, this is not the most efficient representation of the subject strategy. The description length of the model is unnecessarily increased because the same distribution appears as the output assigned to many distinct input vectors. We would like, therefore, to group together similar distributions to eliminate this redundancy.

This consideration directs us to finding a *maximal response-equivalent partitioning* of the input space, where we try to put as many elements as possible into each response-equivalent set. Let us define:

Maximal response-equivalent partition: A response-equivalent partition $\hat{R}$ of $\mathbf{P}$ is said to be maximal iff it maximises

$$\frac{1}{|\hat{R}|} \sum_{c \in \hat{R}} |c| \tag{2.1}$$

over all possible partitions of $\mathbf{P}$.

Maximal response-equivalent partitioning, if done well, can produce the following benefits:

1. We can specify the outputs of M by specifying, for each equivalence class C of $\mathbf{P}$ wrt R, the action distribution summed over the distributions for every $p \in C$, as opposed to specifying a distribution per element of $\mathbf{P}$.

2. If the set of input vectors that fall in an equivalence class of $\mathbf{P}$ can be described by a rule whose length is shorter than the enumeration of every member of that class, we can decrease the description length of the strategy model.

3. The members of a cluster will be similar in the sense that they all evoke the same reactive response from the subject. We therefore expect that the clusters will define classes of inputs that form leaves in the strategy tree and that the set of clusters define the overall strategy of the subject.

4. The action distributions exhibit noise of a non-Gaussian nature. This makes it difficult to model, and thereby eliminate, the noise. Also, the noise can be easily confused with irregularities in the distribution caused by changes in subject strategy. Computing distributions over the equivalence classes eliminates the noise in the action choices. Noisy actions receive low frequency counts and thus will not influence descriptions of strategies.

2.4.1 Using a good similarity measure

The adequacy of a maximal response-equivalent partitioning, for representative or explanatory purposes, depends on the similarity measure Δ used to compare two distributions. For example, we could create a partition of the input space into inputs that occur outside the *minefield*, the region of the plane where the mines are laid out and inputs that occur inside the minefield. The similarity measure in this case might simply compare the distribution of the speed values: one expects that the subject chooses high speeds outside the minefield than inside it. This would be a poor characterisation of the subject strategy because it ignores the goal structure inside the minefield and while approaching it. This structure contains information crucial to understanding subject performance at the task as well as duplicating it.

Finding a good similarity measure requires an understanding of the nature of the action distributions, posing a chicken-and-egg problem: to know enough about the distributions to understand what measure would separate them out, we first need a measure to separate them out so that we can know enough about the nature of the variations among them.

2.5 Local over Global Methods

The aims of finding a near-maximal response-equivalent partition and of doing so with an adequate similarity measure make this problem difficult to solve using global methods. A global method is one which estimates outputs based on information derived from the entire output space. An example of this would be a neural network, trained to fit a non-linear function that maps each input vector in the input space to a value close to its corresponding output vector, while minimising some error function. Such a method is unable to partition the space in the first place, leave alone do so maximally.

Some global methods can partition the space but inadequately. Decision trees, for example, creates axis-parallel partitions, a requirement which severely limits the possible partitions of the data. We show empirically that the best partitions are not axis-parallel, a fact that explains the poor performance of decision trees on this data.

Hence, we expect that the best methods for obtaining representative models will be local ones: given a query vector for which an action has to be decided, we find a subset of the input space that is "close" to this query vector and estimate the output from a distribution calculated from these "neighbors" of the query vector. Also, we expect that using local methods will help in a preliminary "separation" of the distributions. The classes of distributions thus obtained will help in developing a measure that can refine the process of separation. This process can be iterated till we obtain a good classification of the kinds of distributions the data contains.

In the next chapter, we discuss the particular local model that we have developed. This model was motivated by the two peculiarities of analysing the data: the need to perform a partitioning that requires no *a priori* understanding of the subject strategy and the ability to do so without recourse to a similarity measure between action distributions.

Chapter 3

Segmentation and Model Construction

In this chapter, we describe the methods we have developed for changepoint detection and model construction. Both these descriptions require a representation of the subject strategy on the task. Hence, we first describe such a representation. We then present our concept of a *policy derivative* that we use to find changepoints. We discuss the results of our segmentation methods. We then describe in detail the design and implementation of the local model we have developed as a representative model for the task. Local models have two main methods: one that finds the local inputs close to a query input for which a model response has to be computed and one that uses these neighbors to compute a response. We compare two methods used for the latter purpose, weighted averaging and locally-weighted regression (LWR), and discuss a novel extension to LWR that improves performance dramatically. We conclude the chapter with discussion of some empirical extensions of the stateless model that introduce state in small fractions of the input space, making it computationally feasible to use large-dimensional state spaces in order to improve model performance.

3.1 The Lookup Table Representation

The lookup table representation of a subject constructs, from the subject's data run, a table which contains, for each input vector that was encountered in the run, the frequency counts of the actions that were taken in response to it. We choose this model since it is simple to construct online and requires no *a priori* partitioning of the space. On average, on the order of 10^4 entries per 100 episodes of play are generated per subject. About 95% of the entries are associated with a single action and nearly all of the rest with less than 10 actions.

3.2 Segmenting the Input Data

The essential idea is to process episodes in blocks of size w and learn stochastic policies from each of them. Given a distance measure over the space of policies, we can compute the distance between policies over two successive blocks. We study how this distance changes with time (i.e., block number) and use a standard threshold of the mean plus twice the standard deviation on the change in policy distance between successive blocks to identify blocks where there is a significant change in the action policy. More formally, let $\pi_{i,i+w}$ be the stochastic policy derived from episodes $e_i \dots e_{i+w} \in \mathbf{E}$ and let $\pi_{i+w+1-s,i+2w-s}$ be the stochastic policy derived from the next block of w episodes, $e_{i+w+1}, \dots, e_{2w}$. Let Δ_P be a distance function over the space of all policies. Then the *policy derivative* at the block of width w that starts at episode i is

$$\delta(i, w, s) = \frac{\Delta_P(\pi_{i,i+w}, \pi_{i+w+1,i+2w})}{w}$$

3.2.1 A distance measure for lookup tables

For the lookup table representation of policies, a natural distance measure is to compute the average distance between action distributions associated with each p that occurs in the union of the two policy lookup tables. We choose KL-divergence of two discrete distributions as a measure of the distance between them. For lookup tables with 10^4 distinct p's, it is impractical to compute the average distance over all p's. To get around it, we use a Monte-Carlo sampling technique: we compute the average distance over a small number of randomly chosen p's from the policy tables. We experimentally determine that repeatedly and randomly sampling 5% of the p's from each lookup table keeps the variance of the distance measure obtained under 1% of the mean of the distance distribution.

3.2.2 Good values for w and s

Changing the value of s, the amount of overlap between consecutive windows, sharpens the peaks: if s is sufficiently large, the policy derivative is high only if there is a significant change in subject policy, since the two windows have a fair fraction of the actions in common. Also, shifting the second window allows us to generate more points for a given value of w. Naturally, we can choose w to be small to generate more data points and thereby locate changepoints more accurately. However, certain properties of the data favor making w large:

1. **incompleteness:** Since episodes have different configurations, each P_i might only partially represent the complete policy that the human has formulated for the task at t_i. Making w as large as possible allows us to construct policies that span $\mathbf{P}$ better, which, as we shall see, reduces the complexity of calculating the distance between two policies.

2. **noise:** Noise in the data is caused due to two reasons. First, the joystick output does not accurately reflect the decisions made by the subject—even if subject strategy computes fairly precise responses, they must be approximated by the simulation's joystick interface. Second, distraction and fatigue can introduce "policy lapses," which pollute the distributions. Choosing a large w can compensate for these effects by aggregating the distribution over many episodes.

We experimentally determined the best value of w and s for our data to be 20 and 5, respectively.

3.2.3 Results

Figures 3.1 and 3.2 show the results of using this Monte-Carlo sampling technique to rapidly estimate the policy derivative for Subject 3 on the fourth day of training. The derivative is larger in earlier parts of the data run. This is possibly because early on, the subject defines the planning component of his strategy more than the reactive component when learning the game and this changes the policy more dramatically. In contrast, over the latter half, the subject refines the reactive component to improve performance, leading to slighter, subtler, changes in the response distributions.

As a result of these different scales in the derivative plots, we computed our thresholds—the mean plus twice the standard deviation—separately for the two parts. These thresholds each pick 4 to 5 peaks, at approximately episodes 10, 200, 250 and 300 in the first half and episodes 350, 400, 425, 575 and 625 in the second half. Seven of these nine episodes are points close to rising edges in the performance graph between two learning plateaus, where the subject's performance undergoes major improvements, while two mark falling edges. There are shifts in the policy that are not marked by the method: e.g., the two falling edges ending at episodes 400 and 480, approximately.

Hence, our method is sound but not complete. It is, however, consistent with predictions from the cognitive science literature in finding that there are only a few shifts in strategy or policy in human subjects learning this task.

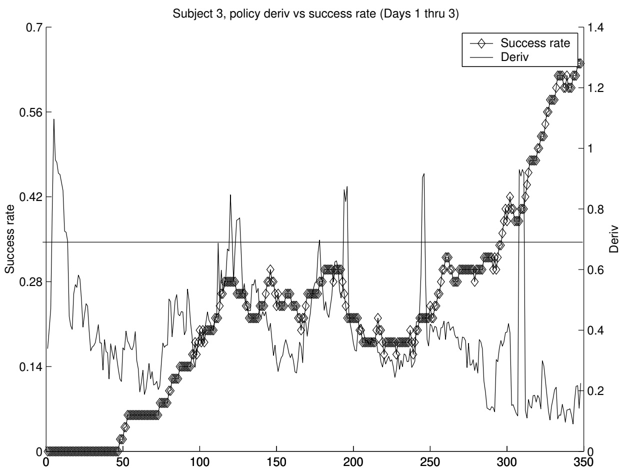

Figure 3.1 : Policy derivative overlaid on success rate (Subject 3, Days 1 to 3)

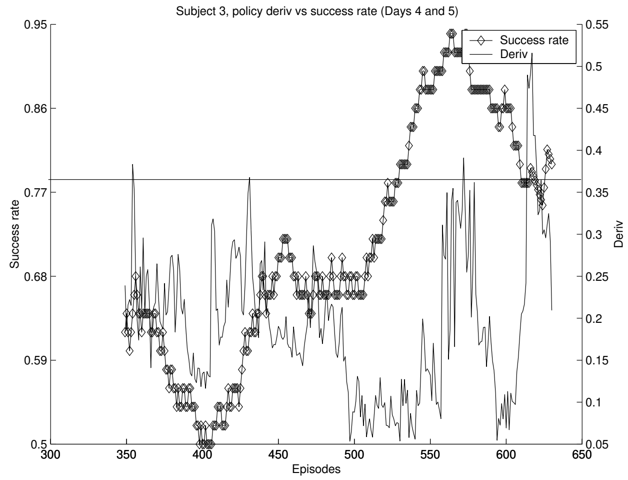

Figure 3.2 : Policy derivative overlaid on success rate (Subject 3, Days 4 and 5)

3.3 The Lookup Table Model

Now that the data stream is segmented into nearly stationary segments, we will learn models that map the perceptual input vectors to distributions over actions using data from each of these segments. In this section, we discuss the *lookup table model* that we have developed for this purpose.

The lookup table model uses the lookup table representation. Given an input vector, p, the model chooses an action according to the action distribution associated with p, if p is available in the lookup table. If an input vector p is not found in the lookup table, the model locates its 100 nearest neighbours using the L_2 norm of the difference between two vectors as the distance function[1] The search is done efficiently by storing the input vectors in a kd-tree [FBF75]. We used ANN [AM98], a program that performs nearest-neighbour searches, which we modified to handle wrap-around dimensions. ANN stores vectors as an array and builds a kd-tree containing indices

[1]The bearing dimension is treated as a *wrap-around dimension.* We first calculate the absolute difference d of the elements and the difference d' of the upper limit of the value of this dimension (12) and d. The term in the L_2 norm for this dimension is chosen as $\min(d, d')$.

into the array. The entire data structure occupies about a megabyte of memory per 100 episodes of play, which is a manageable amount of data for primary memory, making neighbor lookup a fast operation.

We investigated two methods for predicting the action using the distributions associated with the nearest neighbours for a query. Both methods first associate a single action to each neighbor input vector by drawing it from the distribution associated with that vector in the lookup table. Given these 100 candidate actions, the response is calculated using either:

1. **Weighted Averaging:** An action is computed as the weighted average of the candidate actions.

2. **Locally-weighted polynomial regression [MAS97]:** A linear fit of the candidate actions which minimises the weighted sum-of-squares error is computed. The action is chosen using this fit.

In the next section, we address various issues that are common to both response calculation methods (weighted averaging and regression) and discuss details of our implementations of these methods.

3.3.1 Implementing the Local Methods

In both methods, each dimension is scaled so that its lower limit scales to 0 and its upper limit to 1. We used a Gaussian weighting function, so that the weight assigned to a vector at distance d from the query vector is $e^{\frac{-d^2}{2k^2}}$, where k, the *kernel width*, is defined so that a weight of 0.01 is assigned to the neighbour that is some multiple, c, times further from the query than the nearest neighbour. We empirically found that $c = 2$ is a good value, as far as achieving a good fit is concerned.

Our response calculation methods do not always return action responses that are multiples of the step values for turn and speed. These values are presented as is to the simulation, because it will round them off to the nearest legal value, except in two cases: a turn of $2(-2)$ is forced to $4(-4)$ rather than 0 and a speed in $[1, 2]$ is forced to 5 rather than 0. This was done because response calculation picks a small speed far more frequently than the human picks a $\langle 0, 0 \rangle$ action. Hence, we felt that a small turn/speed choice was more representative of a slight move on the human's part, rather than no move at all.

There are two responses to be computed at each step: a turn and a speed. We investigated two methods of calculating the paired response:

1. computing the averaging/regression twice, once for the speed values and once for the turn values, and

2. iterating the following procedure until the same output vector is calculated in two consecutive iterations: calculate the speed and turn separately, as in the first method. Form two *augmented queries*: one with the computed speed value appended to the original query and another with the computed turn value appended to it. Augment each neighbor with the turn value and perform response computation using the turn-augmented neighbors and the turn-augmented query to obtain a new speed value. Calculate a new turn value similarly.

The first method assumes that the actions are not correlated and can therefore be calculated independently. The second method attempts to account for correlation by making an initial, independent guess and comparing it with the response vectors obtained from the lookup table, so that the choice of turn affects the choice of speed and vice-versa. We found that independent of the choice of response-calculation method, correlating the response components improves performance by an average of about 1 percentage point. This performance benefit remained in later extensions to state-based models and so we performed all experiments using correlations.

The lookup table model was simulated on the same minefield configurations that were played by the subject whose data run was simulated. This was achieved by setting the seeds for the random number generators (which places the mines on the fields) to those used during play by the subject. However, this does not mean that the simulation would see precisely those readings that the subject did (thus

Subject	Days				
	1	2	3	4	5
4	109	412	412	607	768
1	88	173	289	417	534
2	105	206	348	510	664
3	59	141	251	365	479
5	83	192	323	484	635

Table 3.1 : Episodes at end of each day

obviating the need for action prediction.) This is because the panel reading values output by the game engine are rounded off to the nearest "step" of the corresponding panel element. Hence, the same response to the same panel readings might produce different resultant panel readings, depending on what the actual underlying value was in each case. Unseen panel readings are also generated because for some input vectors (about 10% of the entries in the lookup table), we have more than one response, which come from different episodes. The response chosen by the model during a test for a certain mine configuration might not be the one that the subject used for that configuration.

3.4 Testing the representative models

To test the performance of our models against that of the subject, we need a standard train/test protocol. Devising a fair protocol for testing prediction accuracies over of a non-stationary data source is extremely difficult. This is because changepoints need to be taken into account for properly formulating the "train" and "test" sets: if the train set is formed from an earlier learning plateau, it cannot be used for constructing a model that is tested on a mine configuration encountered by the subject on a later learning plateau. Fortunately, the day boundaries (Table 3.3.1) are very close to the changepoints in the first four days. We also know from anecdotal

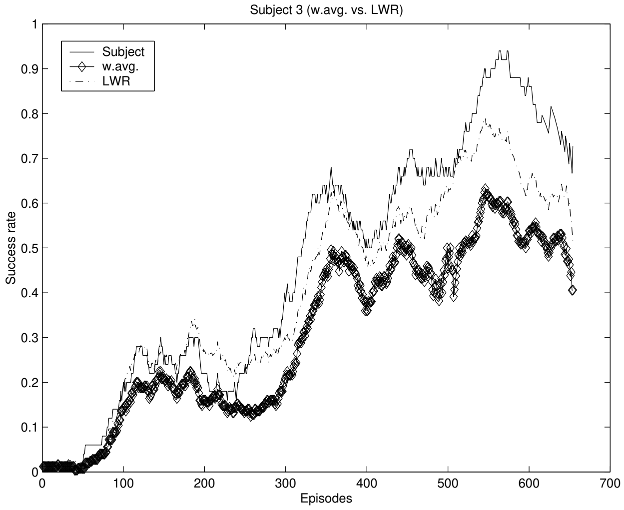

Figure 3.3 : Subject 3, successes, (w.avg. vs. LWR/bde, 0-gram)

evidence that subject performance improves markedly at the beginning of each day, mainly because the subject has had to time to reflect on the lessons of the previous day and is less fatigued than he was on the previous day. Hence, we use a standard cross-validation protocol over each day's data to test the fit of the models to the learning curve of the human subjects. We divide the data for the day into 10 chunks and build a model out of 9 of the chunks and test it on the left out chunk. The process is repeated 15 times and the average performance of the models is reported. We ran this train/test protocol on all of our subjects.

Figure 3.3 compares the performance of Subject 3 to the performance of models constructed from this subject's data run, using weighted averaging and LWR. The fit is near-perfect for the first 4 days (about 500 episodes) but falls off subsequently. We therefore conduct further experiments, and discuss corresponding results, only for day 5 of Subject 3, as this is where we find that our model performs the poorest and hence requires improvement.

3.5 Biased dimension elimination

In Figure 3.3, we find that LWR performs worse than weighted average. This result is counter-intuitive: since the available data are distributed sparsely in the percept space, the nearest-neighbor set is identical for a set of widely-varying queries. Weighted averaging would generate the same response for any such set of queries, while LWR would generate different responses that would vary with a change in the values of the query elements. Our knowledge of subject responses to changes in percepts shows that this variation is monotonic in a local sense: over sets of percept vectors that are close to each other, speed will usually increases when sonar values increase, say. Hence, the linear fit that LWR provides should duplicate this dependence better than weighted averaging does.

We investigated why LWR performed in this counter-intuitive manner by examining the performance of the model as compared to the human at a per-game level. We found that in many situations, LWR computes a low or zero turn or speed value. The percept vector then remains static and the model fails to move, eventually running out of fuel. The low values are computed because the nearest neighbours are all on one side of the query vector in the percept space. Thus, even though the action responses for all the neighbours have high turn and speed values, the predicted response itself is extrapolated to a pair of low values.

To fix this, we introduced the notion of *biased dimension elimination* (BDE). The idea is that if we delete all those dimensions from the neighbour vectors whose values over the neighbours are all greater or all less than the value of that dimension in the query element (and if not all dimensions satisfy this property,) then the regression is guaranteed to interpolate the response than to extrapolate it[2]. Algorithm 1 describes the procedure for deleting the offending dimensions. We conjecture that this procedure might have a cognitive equivalent in that interpolation is more easily performed mentally than extrapolation. Hence, when presented with vectors containing biased dimensions (as defined above), the subject will choose to ignore the

[2]If all dimensions are deleted, we do not apply BDE. This happens less than 5% of the time.

biased dimensions rather than spend time calculating the correct extrapolation.

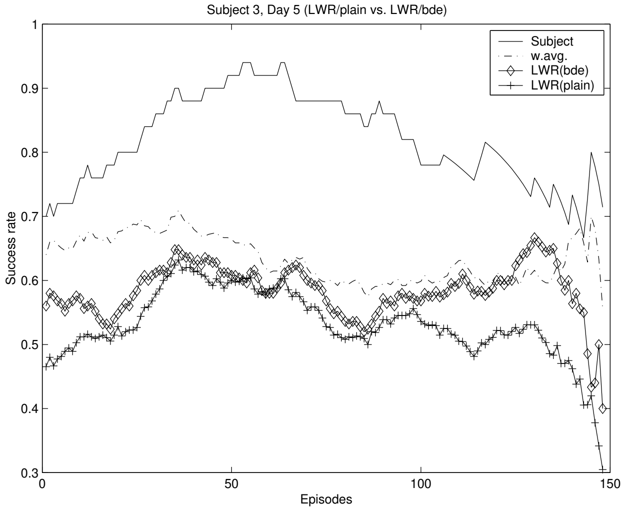

Figure 3.4 : Subject 3, Day 5, successes (LWR/plain vs. LWR/bde)

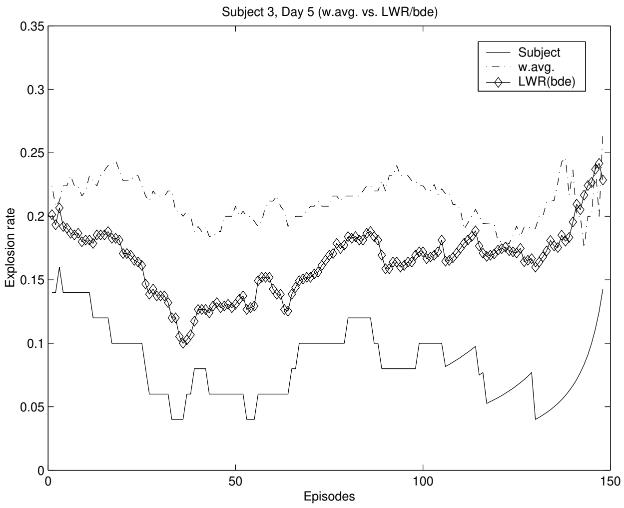

Figure 3.5 : Subject 3, Day 5, explosions (w.avg vs. LWR/bde)

Figure 3.4 shows the improvement obtained by using BDE. Though the model based on regression continues to perform worse than the one based on weighted averaging, the explosions rate of the two methods (Figure 3.5) shows better performance with regression. This encouraged us to continue considering LWR as a possible candidate for unseen-panel response estimation.

The performance gap between the model's and the subject's learning curves point to the need for richer strategy models. Our current models are *reactive*: they do not account for memory in the subject. We intend to model this and, as discussed earlier, the planning component of subject strategy, by extending our models to represent state, the idea for which we present in the next section.

3.6 Richer strategy models

In this section, we discuss our formal model for extensions to the model to handle state. We describe the particular models based on this formalisation that we have implemented and present results for these models.

A state-based model builds its state by memorising past percepts and the associated actions. However, since humans do not have the capability of storing information about the past for an arbitrary length of time, we expect that the subject's current state, at a given point in time, is a function of some finite number of percept-action pairs in the history of interaction with the game up to that point. The current state at time t can be modeled as:

$$s(t) = f(\langle p_t \rangle, \langle p_{t-1}, a_{t-1} \rangle, \langle p_{t-2}, a_{t-2} \rangle, \cdots, \langle p_{t-k}, a_{t-k} \rangle)$$

$p_i \in \mathbf{P}$, the set of percept vectors, $a_i \in \mathbf{A}$, the set of action vectors, denoting that at most k time ticks in the past affect the current state. We call $f(\cdot)$ the *state construction function.*

The use of state can be incorporated easily into the lookup table representation. We can think of $s(t)$ as the current percept and count action frequencies for all instances of $s(t) = f(\langle p_t \rangle, \langle p_{t-1}, a_{t-1} \rangle, \langle p_{t-2}, a_{t-2} \rangle, \cdots, \langle p_{t-k}, a_{t-k} \rangle)$ in the subject's data run. We will refer to this as the *augmented percept.* Since each percept is associated with a distribution of actions rather than a single action, each sequence of k percepts generates multiple augmented percepts: if there are an average of

n actions in the discrete distribution associated with a percept, the number of augmented percepts generated by a sequence of k percepts is n^k. Since a large number of percepts are associated with a single action, n is about 1.5 and for small values of k, augmentation does not create a serious overhead as far as space is concerned. It does, however, intensify the problem of data sparsity, which is a concern for our modeling efforts.

We now discuss the state construction functions $f(\cdot)$ we used and the implications of the results obtained from them.

3.6.1 N-gram models

We first experimented with augmenting percepts with only the past actions. Borrowing terminology from statistical natural language processing, we refer to these models as *n-gram models.* A model that remembers the past n actions is called an n-gram model. Hence, a 2-gram model would be represented by the state construction function:

$$f(\langle p_t\rangle, \langle p_{t-1}, a_{t-1}\rangle, \langle p_{t-2}, a_{t-2}\rangle, \cdots, \langle p_{t-k}, a_{t-k}\rangle) = \langle p_t\rangle\langle a_{t-1}\rangle \tag{3.1}$$

The introduction of actions in the augmented percept requires a different scaling scheme for the speed value. Since the number of possible values for the speed value is only 8, which is less than that of any of the other elements in the 1-gram model input vectors, the nearest-neighbour search tends to pick vectors that have the same speed value regardless of the values of the other elements. This means that once the model chooses a certain speed, it “locks” into that speed for a while, causing an increased number of explosions. To remedy this, the speed values were scaled differently: from 0 to 0.1, rather than to 1.

Figure 3.6 shows the 1-gram model performing better than the 0-gram model. This performance is matched for the other subjects as well. Figures 3.7 and 3.8 show model performances (1-gram, LWR with biased-dimension-elimination) for Alfonzo and Heidi, respectively, who were novice subjects and did not achieve the high levels of accuracy that Ceasar did.

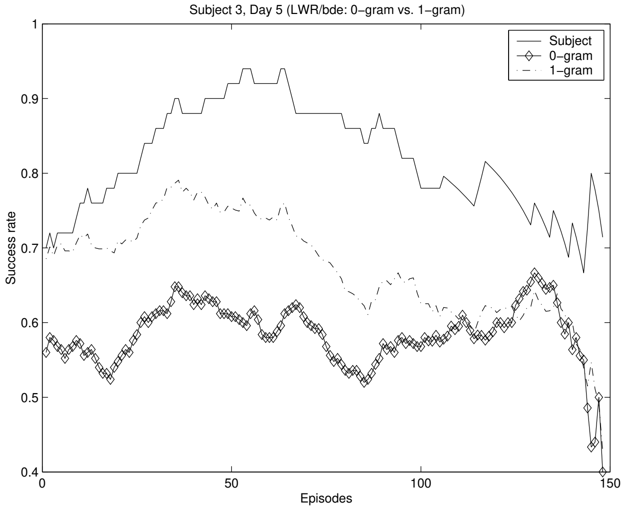

Figure 3.6 : Subject 3, Day 5, successes (0-gram vs. 1-gram, LWR/bde)

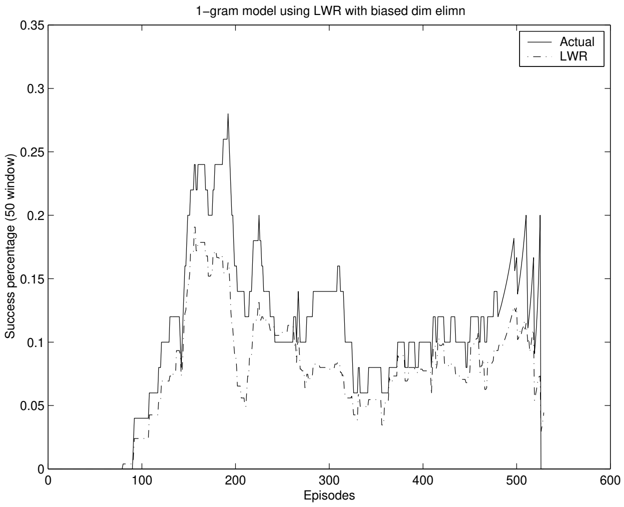

Figure 3.7 : Subject 1, successes (LWR/bde, 1-gram)

3.6.2 The two-tier extension

The main problem with extending state is that it intensifies the problem of data sparsity. Also, the exponential dependence on k, the length of history, increases

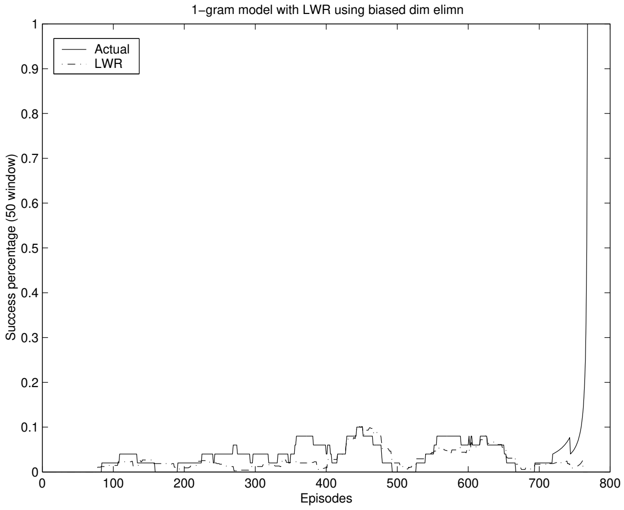

Figure 3.8 : Subject 4, successes (LWR/bde, 1-gram)

Algorithm 1 (**Biased dimension elimination**) Given a set of vectors P and a query q, return the set with dimensions biased wrt q eliminated

for each dimension d **do**
  $P' \leftarrow$ set of points in P that receive a weight $>= .01$
  biased_dim $\leftarrow$ NOT($\exists p_1 \in P'(p_1[d] < q[d])$ AND $\exists p_2 \in P'(p_2[d] > q[d])$)
  level_dim $\leftarrow (\forall p \in P'(p[d] = q[d]))$
  if biased_dim AND NOT level_dim **then**
    Delete the d^{th} entry from all $p \in P$.
  end if
end for
if all dimensions were deleted **then**
  Restore all dimensions.
end if

memory usage ten-fold for $k > 5$. Hence, further extension to state over the entire lookup table was not feasible.

An analysis of model performance showed that extending state wholesale was unnecessary. When compared with subject performance on the time taken to reach a range panel reading of 700, the 1-gram matches the subject even on Day 5. This implies that the performance gap results from incorrect modeling of subject action choices within the minefield. We therefore commenced a more fine-grained analysis of subject strategy inside the minefield by first defining such situations in terms of the values of the panel display. A *close mine* situation, where the submarine is hemmed in by the mines, is one where at least n of the sonar values are greater than k. The complement of a close mine situation is called a *far mine* situation.

The *two-tier model* uses two lookup tables, one each for close mine and for far mine situations. The latter uses the state generation function in Equation 3.1. The former uses the following state generation function:

$$f(\langle p_t \rangle, \langle p_{t-1}, a_{t-1} \rangle, \langle p_{t-2}, a_{t-2} \rangle, \cdots, \langle p_{t-k}, a_{t-k} \rangle) = \langle p_t \rangle \langle p_t - p_{t-1} \rangle \langle a_{t-1} \rangle \langle a_{t-2} \rangle \langle a_{t-3} \rangle \langle a_{t-7} \rangle \tag{3.2}$$

The augmented percept contains the difference between the current percept and the one at the previous time step and the last four actions, taken in time steps of two, rather than one. We explain these choices as follows. The difference in panel readings is useful because the action choice for a close mine situation is quite possibly dictated by how that situation was arrived at. Picking past actions in time steps of more than one is useful because the joystick interface might capture responses from the joystick quicker than the subject can move it. A time step of two attempts to correct for this variation between the “frame rates” of the subject and the model.

When in a close (far) mine situation, the two-tier model performs regression on the data points in the close (far) mine table. This partitioning allows state extension without a toll on memory because close mine situations account for less than 2% of the percepts in the subjects’ data runs.

Figure 3.9 compares the performance of a two-tier model against a one-tier model. Both models use LWR/bde.

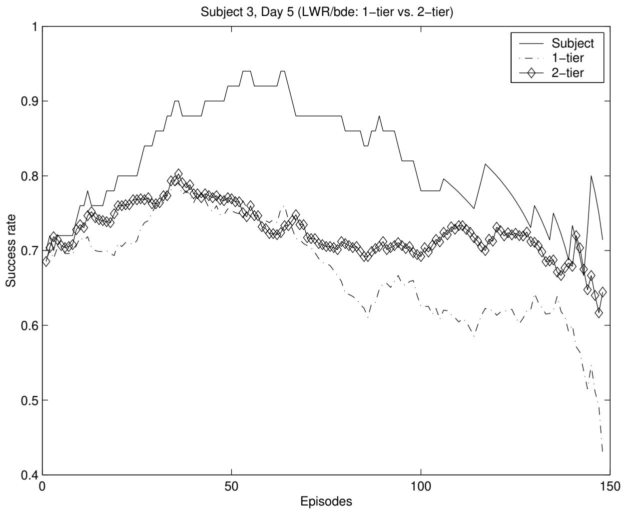

Figure 3.9 : Subject 3, Day 5, successes (1-tier vs. 2-tier)

Chapter 4

Previous Work

In this chapter, we explore the space of modeling methods used for modeling and duplicating human behavior on tasks requiring the construction of complex strategies. We summarise existing literature and compare the performance of one of these approaches, decision trees, with the methods described in previous chapters.

There are two main approaches to strategy modeling discussed here. The first assumes a cognitive architecture and a set of parameters for this architecture that will best duplicate the results obtained from subject experimentation. The architectural parameters that best fit subject performance, and the algorithm used to learn them, are given a cognitive interpretation that explain the evolution of strategy on the task. The second approach, in which category our methods fall, is to use a learning algorithm to infer a percept-action mapping from data. We discuss previous work in these two categories in the sections 4.1 and 4.2, respectively.

4.1 Cognitive architectures

Two main cognitive architectures, SOAR [New90] and ACT-R [And93], have been used extensively in cognitive modeling. Both systems maintain procedural and declarative knowledge about the world. Procedural knowledge contains rules that fire when they find a match on elements in declarative memory. The architectures assume that humans adopt goal-oriented behavior—certain states are desirable and knowledge is used rationally to achieve them—and maintain goal stacks that reflect a certain task decomposition. The manner in which the goal stack is built is governed by model parameters, which can be assigned by humans or learned by the system. The choice of declarative elements and the architectural parameters are decided by assumptions made about the task being modeled. The main conceptual difference in the methods arise from the use and modification of represented knowledge and the handling of the goal stack.

SOAR is supplied with state descriptions of the world and operators for modifying states. It is a parallel rule-firing system that fires every rule that finds a match. Problem solving in SOAR involves finding the right sequence of operators to achieve the goal state and search in a problem space is used to propose and order operators and make declarative memory modifications. The goal stack is expanded in response to an *impasse*—an architecturally-detected inability to use available operator and operator-ordering to change state. SOAR can choose to solve any goal in the stack; solving an earlier goal automatically invalidates all goals closer to the entry point of the stack. The choice of operators and the structure of the states model task assumptions.

ACT-R is supplied with a network of *declarative memory elements* (DMEs) or *chunks*. It is a serial rule-firing system that can only fire *production rules* that match the top-most goal on the stack. Goals are also DMEs. Conflicting rules are not resolved by searching for a good ordering in an operator space; instead, the time taken to instantiate a production rule is decided by the *production strength* of the conditions in the rule and the *activation strengths* of DMEs matched by these conditions. These strengths are architectural parameters of ACT-R, which are assigned by the system's user. Along with the choice and structure of DMEs, which contain *slots* with other DMEs as values, the strength parameters model task assumptions in ACT-R.

[BSR98] models subject performance on the Kanfer-Ackerman Air Controller Task [KA94] with ACT-R. The ACT-R system is provided with rules for landing aircraft depending on aircraft size, runway size and ground conditions. The assumption made by the model is that differences in performance on the ATC task are a result of different encodings of information available about ground conditions and the relative importance given to these encodings, as task conditions change. The parameter space for the strategies discussed in small in comparison to our task:

the probabilities of a small number of encodings are varied to obtain matching performance.

[PHW+93] develops an intelligent control system for a dynamic, unpredictable environment using SOAR, while [LA96] uses it for the *building sticks task* (BST). Though the aim here is not to model human strategy on this task, these give us insight into the assumptions that would have to be made to use SOAR for such modeling efforts. In both tasks, *a priori* assumptions that require fairly detailed knowledge about the task domain have to be made. E.g., the definition of a *problem bias* in the BST and the use of fixed mappings between plane orientations and changes in motion, which have to be encoded manually.

As is seen in these examples, the primary difference between work using ACT-R and SOAR and our methods is that in the former case, it is imperative that some cognitive assumptions be made about the task. This involves, among other things, a particular choice for operators and DME attributes and particular assumptions about the parameter space that has to be explored. Our task is too complex to allow for such assumptions to be explored adequately for an optimal choice.

4.2 Learning algorithms

Learning algorithms in AI are traditionally classified into two main categories([Mit97]):

1. supervised learning, where each training instance is labeled with the expected output or the result, for that input, of the *concept* to be learned, and

2. unsupervised learning, where the results of the concept for the training instances is unknown.

It is important to understand what components of the human learning on our task are in which category, so that we look in the appropriate subspace of the space of all learning algorithms. Partitioning the input space is a supervised learning problem because the label for each input is the action, which is known. Segmenting the space is unsupervised. Each episode has a hidden label, which is the learning plateau to which it belongs. Analysing the human responses as a non-stationary time series is intended to reveal these labels. Previous work has assumed that subject data is stationary, thus concentrating on the supervised learning task.

4.2.1 Modular neural networks

In [ADP00], the task in question is the piloting of an airplane on a fixed "highway in the air." As the authors state, this is a purely reactive task and doesn't require the pilot to maintain state or compute navigational maps or strategies from incomplete inputs. Hence, modular neural networks are ideally suited for this task. In contrast, our task not only requires the human to deduce maps but also to devise high-level strategies for navigating those maps. Additionally, our aim is to build explanatory models online: neural nets are notoriously difficult to train quickly and to interpret. Our experiments with training modular neural networks on this data resulted in large error rates and high convergence times.

4.2.2 Decision trees

Decision trees are a natural choice for explanatory models. In C4.5 [Qui92], the well-known implementation of decision trees by Quinlan, a decision tree can be converted into a set of conjunctions, each conjunction defining an action to be taken if all its conjuncts evaluate to true. This set then represents the strategy of the expert and is easily converted to a verbal explanation by writing the English equivalents of the if-then statements. However, unless this set is small, it is useless for the purpose of communicating the rules to a novice.

We tested decision trees on Subject 3's Day 5 data. For this experiment, we modified C4.5 to function as a decision tree library rather than in the interactive mode that is otherwise available in the code. We make a classification request for a query and obtain the distribution of actions stored at the leaf node that matches this query. The action choice is drawn from this distribution. The experiments were

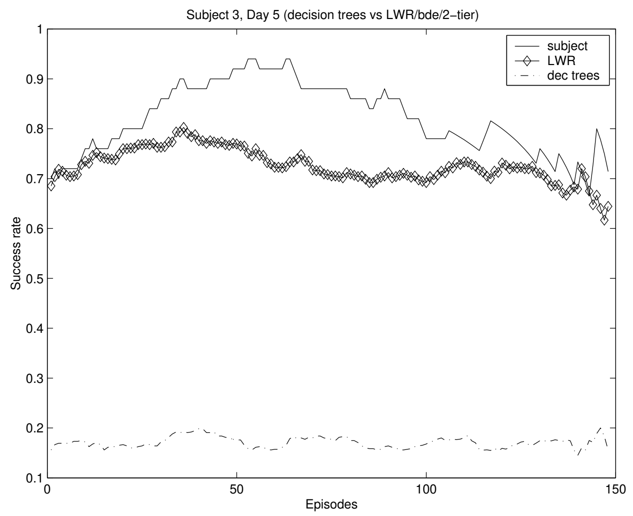

Figure 4.1 : Comparison of LWR/BDE with decision trees (Subject 3, Day 5)

conducted with the same parameters as the 2-tier model. Two decision trees were trained, one on a 1-gram lookup table and the other on the table constructed using the state construction function in Equation 3.2. In both cases, the attributes were marked as "discrete." For each lookup table, two trees were trained, one to output a turn classification and another to output a speed classification. The decision trees performed poorly (Figure 4.1) as performance models.

The trees failed to be explanatory models, as well. They were extremely large and couldn't be converted into a concise set of rules. These performance figures, namely, the success rate and the size of the trees, do not improve if we attempt to prune the tree or if we mark the attributes as "continuous." In the former case, the tree does not reduce significantly in size, even for low confidence thresholds for pruning.

The poor performance of decision trees on this data highlights the inability of global methods to model performance on complex, multi-component tasks. The information gain criterion used to learn decision trees is ill-suited for our data. A typical split oscillates between two extremes: either there is no single attribute better than all others at separating the distributions, giving nearly equal gain values across attributes, or there is a strong but false correlation between the classification and one of the attributes that generates incorrect classifications. An example of the latter case is the strong positive correlation between the previous and the current actions. This frequently locks the model into a single action choice, resulting in time-consuming "looped" movements or in a quick dash into mines. Moreover, the split information ratio doesn't add information: each attribute generates on the order of the same number of outcomes, which usually share the data points uniformly.

[SHKM92] studies the induction of decision trees using data generated by expert pilots. This study attempts to develop a control system quickly by using a human as a tutor and hence, the focus is on modeling a human strategy that has already stabilised rather than on understanding the path of evolution the human travels to arrive at it. Again, the task chosen is a highly reactive one, whereas ours, to reiterate, is not. Also, the data is preprocessed using domain knowledge to prevent the most-common class from biasing the induction, which violates the basic principle of our approach, which is to use only automatic methods both for dealing with noise and for partitioning the input data.

Though in this and the modular networks studies, there is mention of using the induced trees or nets as explanations of human strategies, the authors have the advantage of dealing with fairly stable data and with fixed trajectories. First, our data is non-stationary, which means that we have a difficult, unsupervised learning task to first solve that generates the stable plateaus to which we can apply model-building. Second, our data is gathered for trajectories which are generated at random and which are different for each episode. The data is a mixture of distributions that are distinguishable only along non-axis-parallel partitions of the percept-space. This, and the presence of non-Gaussian noise, explains the poor performance of the global methods we tested on this data.

Chapter 5

Conclusions

5.1 Summary

This thesis has presented a data-driven method for producing action choice policies on a complex, multi-component visualmotor task that are behaviorally-equivalent with respect to human performance on this task. Our methods are capable of adapting to individual differences in the evolution of learning skills on this task. The task we have analysed is the NRL Submarine Navigation Task. We have argued that the peculiar properties of the data generated by this task—noisy, sparse, high-dimensional and non-stationary—make global methods unsuitable for modeling it. This is because subject strategy is encoded in the response distributions in the data. The percept space contains a mixture of these distributions that cannot be separated by traditional global methods for inference or fitting, like decision trees and neural networks. We handle these mixtures by using "local methods" instead, which dynamically vary the importance of data points in the percept space with respect to the query percept in calculating a response. These methods are based on a simple lookup-table representation of subject strategy that can be constructed in real-time.

We have investigated two methods for calculating responses, given a query and weights for percept data point that denote their importance with respect to that query. The first is weighted averaging and the second is locally-weighted regression (LWR). We have presented a novel extension called *biased dimension elimination* to the latter method that improves its performance by allowing the regression to interpolate a response to the query rather than extrapolate from the available data points.

We have extended the basic, *stateless* model—where only the percept vector is used to calculate a response and no internal state is maintained about responses generated for previous queries—to a richer policy model that incorporates state. We have presented a framework for the construction of richer models and experimented with different extensions based on this framework. We demonstrate that maintaining state is necessary and that high-dimensional state augmentation can be achieved without prohibitive memory costs. We show empirically that such augmentation is required only for a small subspace of the percept space—the "close-mines" space—which denote a region where the subject is hemmed in by mines. This space is defined by an intuitively obtained definition. It is effective in demonstrating the feasibility of and need for augmentation. However, because of its ad-hoc nature, it does not adapt to, and therefore does not duplicate, the performance levels of all subjects at all points in their training period.

5.2 Future Work

5.2.1 Explanatory models

The chief aim of future research on the modeling of this task is to construct *explanatory models*: models of strategies that have a *compact* representation, in the sense that they provide a description of the strategy that is easily understood by humans. The lookup table model, in contrast, is a *representative model*: a behaviorally-equivalent model that need not be compact and need not have any cognitive plausibility. We believe that the methods developed for this model have helped us in a basic understanding of the components of strategy on this task and of the nature of strategic shifts. This understanding will be invaluable in the future development of explanatory models for the task.

Preliminary experiments have shown [Sub00] that the hierarchical graph structure of subject strategy described in Chapter 2 is a valid framework for strategy models. A coarse abstraction of the percept space partitions it by the range reading r into regions where $r > 8$, $4 \leq r < 8$ and $r < 4$ (in terms of the units used to measure the range reading.) This roughly correlates to the regions before, in and after the minefield. Markov models of action sequences in these regions were trained for data collected from before and after a strategic shift on Day 3. Each state of the Markov models can output one of 6 actions, corresponding to a two-way choice of speed ($s = 0, s > 0$) and a three-way choice of turn ($t < 0, t = 0, t > 0$). The output and transition probabilities are before the shift are significantly different from the ones learned after it. These probabilities are also cognitively plausible: they allow an easy and intuitive explanation of the strategic shift.

5.2.2 Automatic partitioning techniques

The Markov model was constructed with an ad-hoc partitioning of the percept space that performs a very coarse decomposition of subject strategy. The partitioning is response-equivalent (Section 2.4) only under a fairly relaxed similarity measure. The model is not representative: keeping track of only the action sequence does not generate action predictions that duplicate subject performance. We intend to explore automatic partitioning techniques that produce partitions for which more detailed explanatory, and, hopefully, representative, models can be built. We conjecture that these two properties of models—explanatoriness and representativeness—are exclusive to some degree. Compactly written models might entail a minimum sacrifice of behavioral equivalence.

Making partitioning feasible might require that we find a basis function that will require fewer parameters to describe percept-response mapping than a simple lookup-table mapping. The search for a response-equivalent partitioning will then be quicker. Possible candidates for such a basis function are wavelets and fractals.

5.2.3 State augmentation

The state construction functions in Chapter 3 were constructed by manual inspection and analysis of model performance. We need methods that will find optimal transformations, and the subspaces of the percept space which require such transformations, automatically.

We believe that the gap in performance remaining even after the state extensions made for the 2-tier model will be handled mainly by better partitioning and state construction. These, along with a better understanding of the explanatory-representative tradeoff are the main challenges faced in future analysis of the visualmotor task data.

Bibliography

[ADP00] C. Anderson, B. Draper, and D. Peterson. Behavioral cloning of student pilots with modular neural networks. Technical report, Colorado State University, 2000.

[AM98] S. Arya and D. Mount. Approximate Nearest Neighbor search (ANN). http://www.cs.umd.edu/~mount/ANN/, 1998.

[And93] J. Anderson. *Rules of the Mind.* Erlbaum, Hillsdale, NJ, 1993.

[BSR98] B. Best, C. D. Schunn, and L. M. Reder. Modeling adaptivity in a dynamic task. In M. A. Gernsbacher and S. J. Derry, editors, *Proceedings of the Twentieth Annual Conference of the Cognitive Science Society*, volume 20, pages 144–159, Mahwah, NJ, 1998. Erlbaum.

[FBF75] J. Friedman, J. L. Bentley, and R. A. Finkel. An algorithm for finding best queries in logarithmic time. Technical Report 75-482, Stanford University, July 1975.

[GS96] D. Gordon and D. Subramanian. Comparison of action selection learning methods. In *Proceedings of the Third International Workhop on Multistrategy Learning, Harpers Ferry*, pages 95–102, Fairfax, VA, 1996. George Mason University.

[KA94] P. L. Kanfer and R. Ackerman. Kanfer-Ackerman Air Traffic Controller task(c) CD-ROM database, data collection program, and playback program, 1994. Office of Naval Research, Cognitive Science Program.

[LA96] M. C. Lovett and J. R. Anderson. History of success and current context in problem solving: Combined influences on operator selection. *Cognitive Psychology*, 31(2), 1996.

[MAS97] A. W. Moore, C. G. Atkeson, and S. A. Schaal. Locally weighted learning. *AI Review*, 11:11–73, 1997.

[Mit97] T. Mitchell. *Machine Learning.* McGraw Hill, 1997.

[New90] Allen Newell. *Unified Theories of Cognition.* Harvard University Press, Cambridge, MA, 1990.

[PHW+93] D. J. Pearson, S. B. Huffman, M. B. Willis, J. E. Laird, and R. M. Jones. A symbolic solution to intelligent real-time control. *Robotics and Autonomous Systems*, 11:279–291, 1993.

[Qui92] J. R. Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, 1992.

[SHKM92] C. Sammut, S. Hurst, D. Kedzier, and D. Michie. Learning to fly. In *Ninth International Conference of Machine Learning*, pages 385–393, Aberdeen, 1992. Morgan Kaufmann.

[SS01a] D. Subramanian and S. Siruguri. Modeling human learning on a visual-motor task. In *CogSci*, 2001. Submitted.

[SS01b] D. Subramanian and S. Siruguri. Tracking the evolution of learning on a visual-motor task. In *International Conference of Machine Learning*, 2001. Submitted.

[Sub00] D. Subramanian. Inducing hybrid models of task learning from visual-motor data. In *Proceedings of the Twenty Second Annual Meeting of the Cognitive Science Society*, Mahwah, NJ, 2000. Erlbaum.